# Documentation for string2.h and string2.c

Steven Andrews, © 2003
See the document "LibDoc" for general information about this and other libraries.

```
#define STRCHAR 256

int strisnumber(char *str);
int okname(char *name);
char *strrpbrk(char *cs,char *ct);
char *StringCopy(char *s);
unsigned char *PascalString(char *s);
char *EmptyString();
char *StrChrQuote(char *cs,char c);
int StrChrPQuote(char *cs,char c);
int StrrChrPQuote(char *cs,char c);
int strreadni(char *s,int n,int *a,char **endp);
int strreadnf(char *s,int n,float *a,char **endp);
int strreadns(char *s,int n,char **a,char **endp);
char *strnword(char *s,int n);
char *strnword1(char *s,int n);
int wordcount(char *s);
int stringfind(char **slist,int n,char *s);
int strchrreplace(char *str,char charfrom,char charto);
```

Requires: `<stdio.h>`, `<stdlib.h>`, `<string.h>`, `<ctype.h>`

Example program: `SpectFit.c`, `LibTest.c`

Most code written 1/99; moderate testing. Works with Metrowerks C. Transferred code from Utility.c library to newly created string2.c library 11/01. Added array reading and `strnword` and transferred to Linux 11/01. Added `wordcount` 1/24/02. Added `StrrChrPQuote` 3/29/02. Added `StrChrPQuote` 10/29/02. Added `stringfind` 1/19/03. Added `strrpbrk` 6/11/03. Added `strreadns` 1/16/04. Renamed `isnumber` to `strisnumber` 6/9/04 to avoid a name collision with some other command. Added `strchrreplace` 2/13/06.

This library complements the standard string library with several useful functions.

Functions and definitions

`STRCHAR` is defined because it is often easiest for all strings to have the same length. That way concatenations and other manipulations are fairly easy. Of course, it doesn't have to be used.

`strisnumber` returns 1 if the string is a number and 0 if it isn't. Any type of number recognized by `strtod` (stdlib library) is recognized as a number.

`okname` returns 1 if the input string is valid as a variable name. The rule is that the first character must be a letter and subsequent characters may be letters, numbers, or an underscore. The length of the string is not considered.

`strrpbrk` returns a pointer to the last occurrence in string `cs` of any character of string `ct`, or `NULL` if not present. It is identical the the standard library function `strpbrk`, except that it is for the last rather than first occurence.

`StringCopy` takes in a string and returns a copy of it. Exactly enough memory is allocated for the copy to contain the entire string; it returns `NULL` if memory allocation failed. This memory should be freed when it is no longer being used with the stdlib `free` function.

`PascalString` is identical to `StringCopy`, except that it returns a pascal type string. The first character is the number of letters in the string. A previous implementation (pre-11/01) added a terminating `'\0'` as well, as with C type strings; this character is no longer added.

`EmptyString` returns a blank string of `STRCHAR` characters, all initiallized to `'\0'`.

`StrChrQuote` is just like the `strchr` function in the ANSII string.h library, in that it returns a pointer to the first occurance of `c` in `cs`, or `NULL` if not present. However, it ignores any `c` characters after an odd number of `"` marks (i.e. within quotes).

`StrChrPQuote` is similar to `StrChrQuote`, but different. It looks for the first occurance of `c` is `cs`, returning its index if found. It ignores any `c` characters in double quotes or inside parentheses. Any level of paranthesis nesting is permitted. If mismatched parentheses are encountered before a valid `c` is found, –2 is returned; if quotes are mismatched, –3 is returned; if no `c` was found, –1 is returned. It is impossible to search for a quote symbol, and the method of preceding a quote with a backslash to make it a symbol rather than a quote, is not supported.

`StrrChrPQuote` is like `StrChrPQuote`, except that it returns the last occurance of `c`.

`strreadni` reads up to the first `n` integers from the string `s`, delimited by white space; leading white space and multiple spaces between integers are ignored. Results are put in the integer vector `a`, which is assumed to be allocated to be sufficiently large. The function returns the number of integers parsed. Any unconverted suffix is pointed to by `*endp`, unless `endp` is `NULL`.

`strreadnf` is identical to `strreadni`, except that it reads floats rather than integers.

`strreadns` is identical to `strreadni`, except that it reads words rather than integers. It is assumed that each string in the list of strings `a` has already been allocated to be sufficiently large to hold the respective word, as well as a terminating `'\0'`. Any strings in `a` in addition to those that were parsed are not modified.

`strnword` returns a pointer to the `n`'th word in `s`, where a word is defined as any collection of non-whitespace characters. It returns `NULL` if there are less than `n` words in the string, and `s` if either `n` is 0 or if `n` is 1 and the first word starts at the left edge of `s`.

`strnword1` is similar to `strnword`, except that it counts words based on the first word starting at the beginning of the string and each subsequent word separated by a single space or tab from the preceding one (other whitespace characters are considered to be part of the word). Thus, a double space implies the existance of an empty word between the spaces. If there is no `n`'th word, either because it is empty or because the string has less than `n` words, the routine returns `NULL`.

`wordcount` counts and returns the number of words in a string, where a word is defined as a contiguous collection of non-whitespace characters.

`stringfind` locates string `s` in an array of strings called `slist`,which extends from index 0 to `n-1`. If an exact match for `s` is found, its index is returned; otherwise `-1` is returned.

`strchrreplace` searches string `str` and replaces all characters that are `charfrom` with `charto`. It returns the number of replacements that were made.